... we gave a mouse an NDK

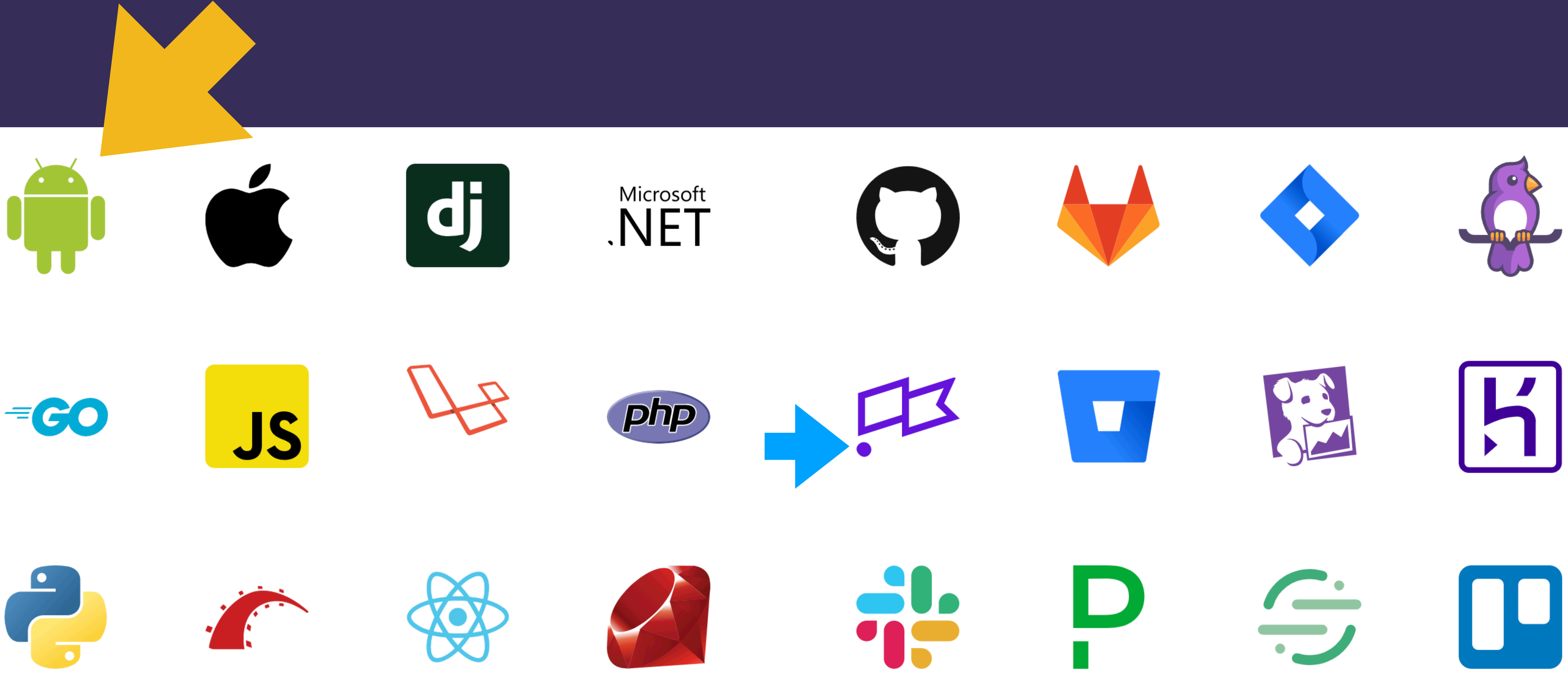
some non android developers' experience with NDK

Armin Ronacher
Director of Engineering, Sentry
@mitsuhiko

Bruno Garcia
Senior Software Engineer, Sentry
@brungarc

SENTRY

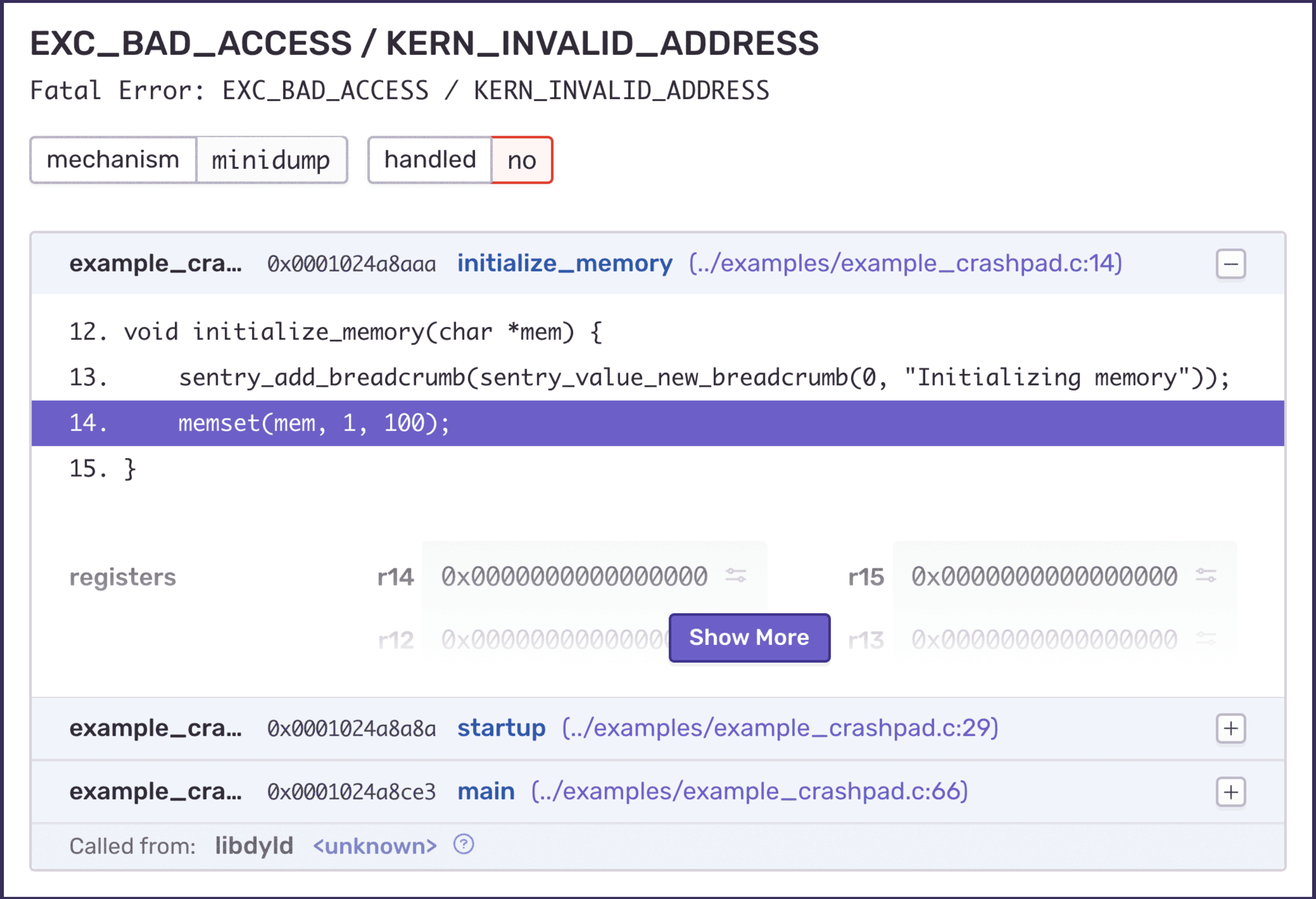
our NDK experience was a bit of an unexpected rabbit hole


dj
Microsoft .NET
GO
JS
php
P

let's talk about us

SENTRY

we're a stack trace company


EXC_BAD_ACCESS / KERN_INVALID_ADDRESS

```
Fatal Error: EXC_BAD_ACCESS / KERN_INVALID_ADDRESS
```

mechanism `minidump` handled no

example_cra... 0x0001024a8aaa **initialize_memory** (../examples/example_crashpad.c:14)

```
12. void initialize_memory(char *mem) {
13.     sentry_add_breadcrumb(sentry_value_new_breadcrumb(0, "Initializing memory"));
14.     memset(mem, 1, 100);
15. }
```

registers

```
r14 0x0000000000000000    r15 0x0000000000000000
r12 0x000000000000000     r13 0x0000000000000000
```

example_cra... 0x0001024a8a8a **startup** (../examples/example_crashpad.c:29)

example_cra... 0x0001024a8ce3 **main** (../examples/example_crashpad.c:66)

Called from: libdyld <unknown>

SENTRY

Armin Ronacher

SENTRY

Director of Engineering

@mitsuhiko

Python & Rust Developer

Bruno Garcia

SENTRY

Senior Software Engineer

@brungarc

.NET Developer

SENTRY

what do we have to do with Android anyways?

You probably know Android better than we do

But we know quite a few things about crash reporting

The goal: stack traces for C, C++, Java, Kotlin, …

SENTRY

NDK

SENTRY

What NDK is

NDK gives us native (C/C++/etc.) code on Android

It interacts heavily with the JVM (ART) via JNI

Android NDK's environment is Linux-ish

NDK Components

What's it based on:

Bionic for libc

some hand picked common libraries (zlib)

we already did Java, we already did C++, …

but we didn't do NDK.

Production Crash Reporting

SENTRY

Production Crash Reporting is Fighting a Paradigm

Production Crash Reporting

Performance and debuggability are often at odds

The lower level the language, the higher the disparity between debug and production build performance

The performance gains come at cost of debuggability

SENTRY

production is all that matters

(for us)

SENTRY

Production on Android

SENTRY

The Runtimes

SENTRY

“Java Runtime”
&
“C Runtime”

SENTRY

Java Runtime

Android Runtime

Runs via some layers of indirection Java bytecode.
Resembles mostly what you get on a traditional JVM.

Specifically you get stack traces from the runtime system from every exception thrown

C Runtime

Very low level, bare minimums.

Interactions with Java via JNI

No native support for producing useful stack traces, dozens of different unwinders for Android non built-in that are good.

SENTRY

Stack Traces

SENTRY

Readable Java Stack Traces

Proguard/R8 obfuscation make stack traces unreadable

Mapping files can be used to resolve method names in stack traces back to the original names.

Readable C Stack Traces

A whole different ballpark.

DWARF information is generally used to restore location information and method names in stack traces once we have them

To get them in the first place is tricky

turning numbers and funny strings into stuff humans can comprehend

Java is easy because Java stack traces are good

Proguard mappings:

```
a.b.c:2 -> was.WeirdThing.method
```

SENTRY

class name:	a.b.C	->	io.sentry.FooBar
method name:	a	->	doSomeFoo
line number:	42		

Preventing Obfuscation

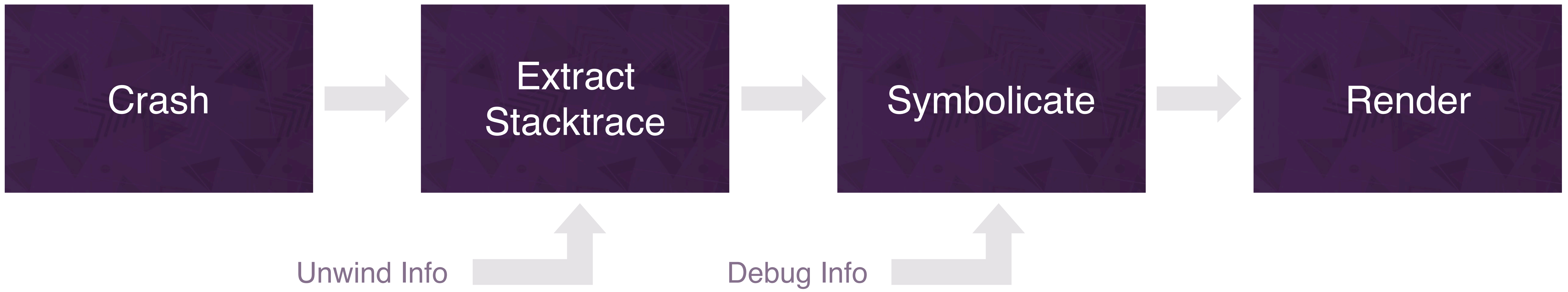
```
-keep public class * extends java.lang.Exception
-keep class com.example.myapp.MyBridge { *; }
```

SENTRY

But C …

How do we get a stack trace?

SENTRY


Crash
Extract Stacktrace
Symbolicate
Render
Unwind Info
Debug Info

SENTRY

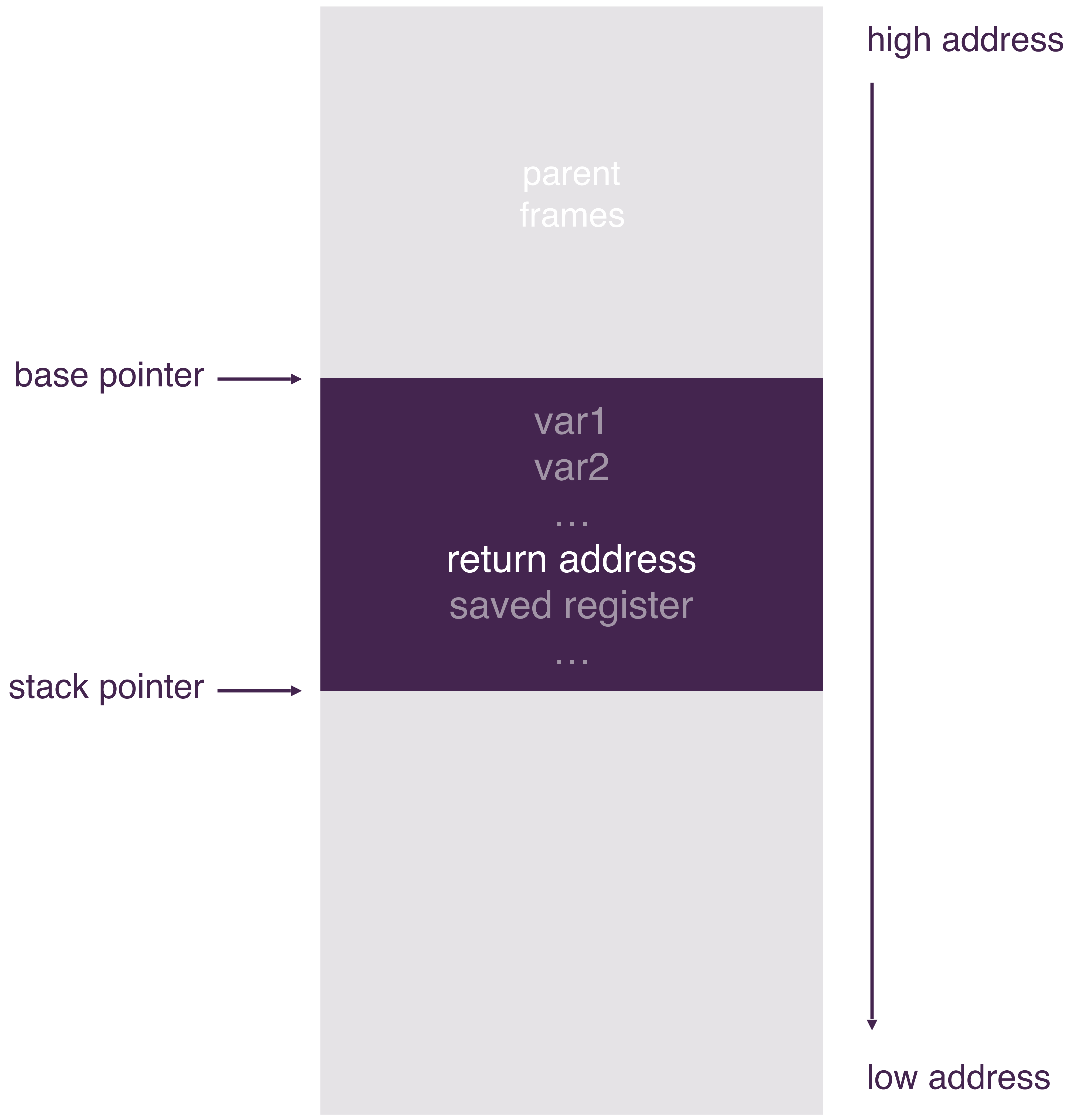
github.com/getsentry/symbolicator

stack walk or memory dump?

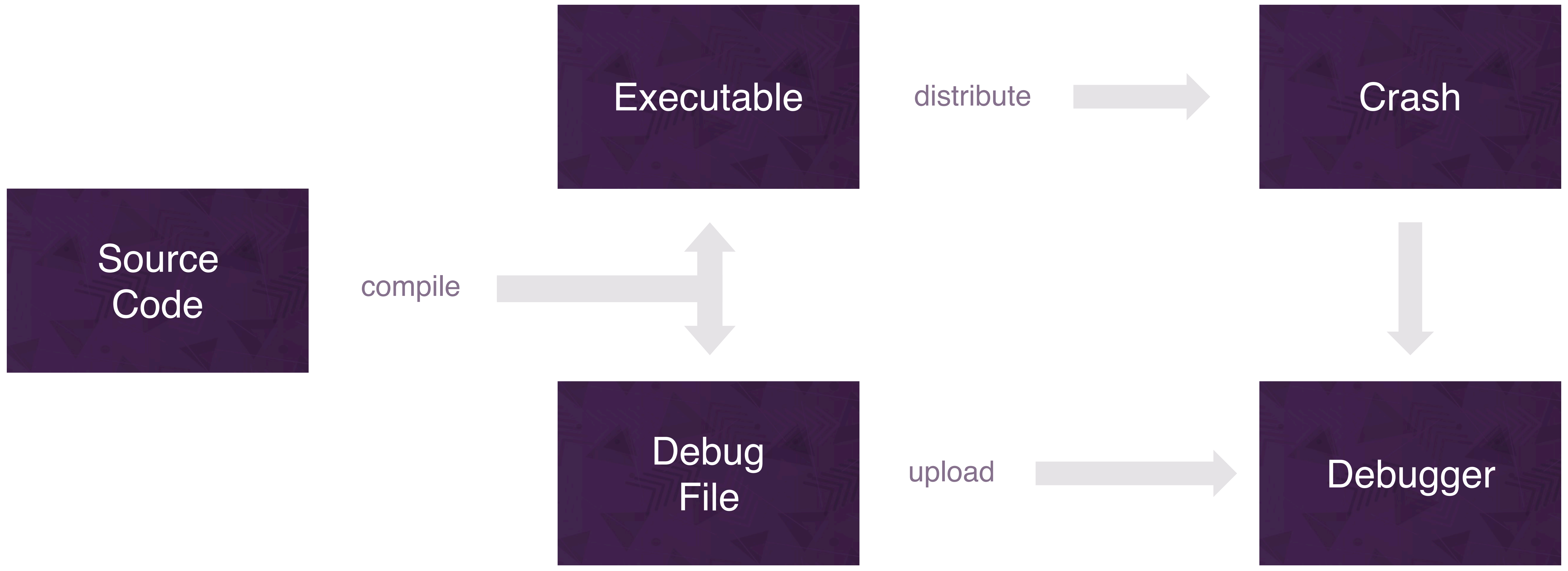
SENTRY

the problem of unwinding

SENTRY


high address
parent
frames
base pointer
var1
var2
...
return address
saved register
...
stack pointer
low address

SENTRY



Source Code
compile
Executable
Debug File
distribute
Crash
upload
Debugger

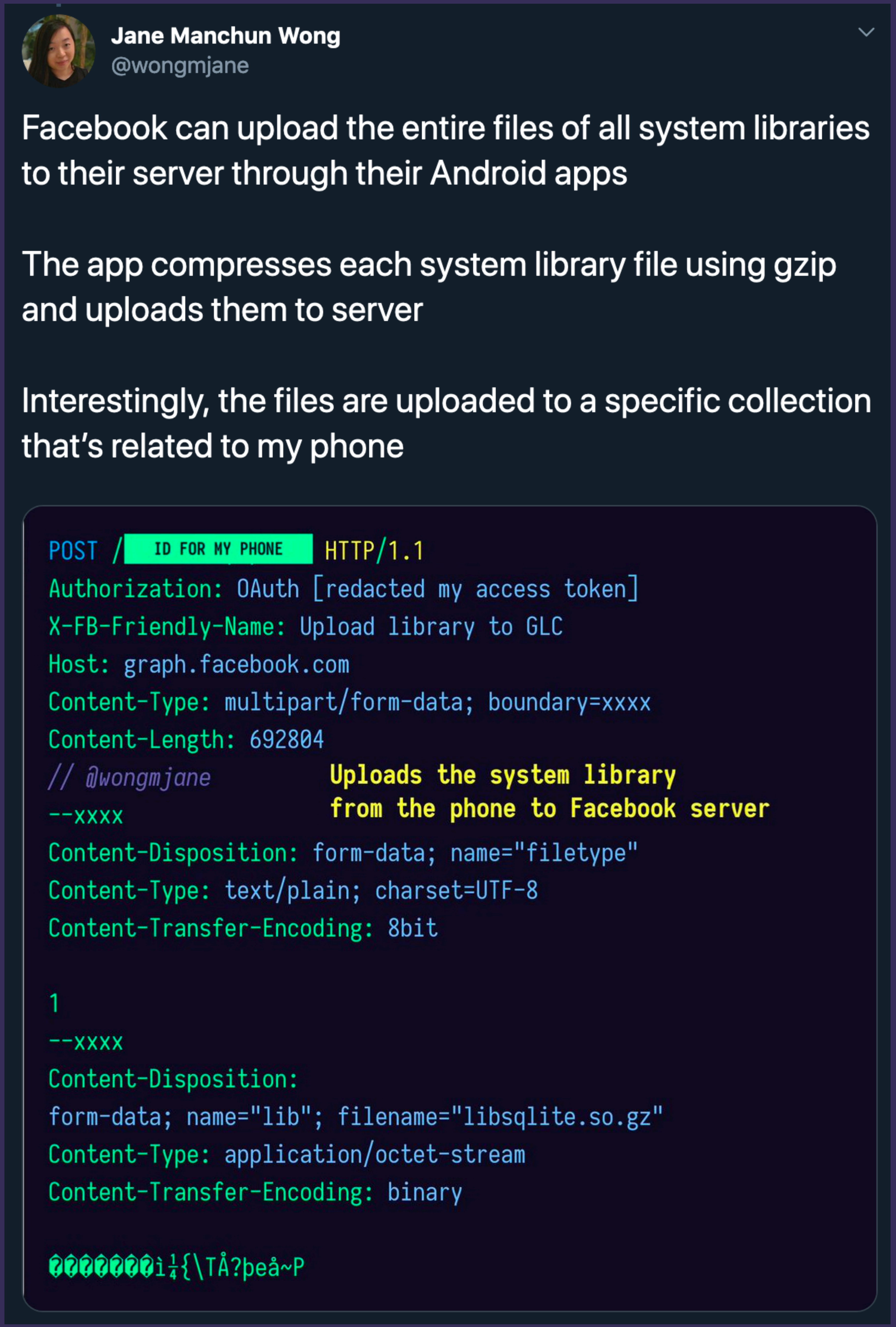
SENTRY

unwinding memory dumps

SENTRY

SENTRY

SENTRY



Jane Manchun Wong
@wongmjane

Facebook can upload the entire files of all system libraries to their server through their Android apps

The app compresses each system library file using gzip and uploads them to server

Interestingly, the files are uploaded to a specific collection that's related to my phone

```
POST /ID FOR MY PHONE HTTP/1.1
Authorization: OAuth [redacted my access token]
X-FB-Friendly-Name: Upload library to GLC
Host: graph.facebook.com
Content-Type: multipart/form-data; boundary=xxxx
Content-Length: 692804
// @wongmjane           Uploads the system library
--xxxx                  from the phone to Facebook server
Content-Disposition: form-data; name="filetype"
Content-Type: text/plain; charset=UTF-8
Content-Transfer-Encoding: 8bit


1
--xxxx
Content-Disposition:
form-data; name="lib"; filename="libsqlite.so.gz"
Content-Type: application/octet-stream
Content-Transfer-Encoding: binary

�������ì¼{\TÅ?þeå~P
```

SENTRY

okay … so what can we do?

SENTRY

stack walk on device

SENTRY

stackwalkers

libcorkscrew
- deprecated, 32bit only

libunwind
- deprecated, google provides android patches

libunwindstack
- C++ monstrosity, actively maintained

libunwindstack

requires custom patches to compile with NDK

requires large sigaltstack to not overflow the stack in the signal handler

development in android master deviated from most NDK compatible forks

gief stackwalker

android can already stackwalk (see ndk-stack)

why is the stack walker not exposed to us?

SENTRY

build id and image addresses

now we need the GNU build id and the image offset for each loaded executable / dynamic library

normally one would use dl_iterate_phdr

this one is missing on older NDKs,

Workaround: parse /proc/self/maps

```
00400000-0040b000 r-xp 00000000 08:01 36                         /bin/cat
0060a000-0060b000 r--p 0000a000 08:01 36                         /bin/cat
0060b000-0060c000 rw-p 0000b000 08:01 36                         /bin/cat
0161f000-01640000 rw-p 00000000 00:00 0                          [heap]
7f01ec015000-7f01ec1d3000 r-xp 00000000 08:01 48677          /lib/x86_64-linux-gnu/libc-2.19.so
7f01ec1d3000-7f01ec3d3000 ---p 001be000 08:01 48677          /lib/x86_64-linux-gnu/libc-2.19.so
7f01ec3d3000-7f01ec3d7000 r--p 001be000 08:01 48677          /lib/x86_64-linux-gnu/libc-2.19.so
7f01ec3d7000-7f01ec3d9000 rw-p 001c2000 08:01 48677          /lib/x86_64-linux-gnu/libc-2.19.so
7f01ec3d9000-7f01ec3de000 rw-p 00000000 00:00 0
7f01ec3de000-7f01ec401000 r-xp 00000000 08:01 48672          /lib/x86_64-linux-gnu/ld-2.19.so
7f01ec46a000-7f01ec5f3000 r--p 00000000 08:01 9746           /usr/lib/locale/locale-archive
7f01ec5f3000-7f01ec5f6000 rw-p 00000000 00:00 0
7f01ec600000-7f01ec601000 r--p 00022000 08:01 48672          /lib/x86_64-linux-gnu/ld-2.19.so
7f01ec601000-7f01ec602000 rw-p 00023000 08:01 48672          /lib/x86_64-linux-gnu/ld-2.19.so
7f01ec602000-7f01ec603000 rw-p 00000000 00:00 0
7ffd808de000-7ffd808ff000 rw-p 00000000 00:00 0              [stack]
7ffd80950000-7ffd80953000 r--p 00000000 00:00 0              [vvar]
7ffd80953000-7ffd80955000 r-xp 00000000 00:00 0              [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0      [vsyscall]
```

SENTRY

sigaltstack / async safety

SENTRY

```
static const size_t SIGNAL_STACK_SIZE = 65536;
stack_t g_signal_stack;

g_signal_stack.ss_sp = malloc(SIGNAL_STACK_SIZE);
g_signal_stack.ss_size = SIGNAL_STACK_SIZE;
g_signal_stack.ss_flags = 0;
sigaltstack(&g_signal_stack, 0);
```

all we want is a symbol server

SENTRY

Putting it Together

SENTRY

NDK side

sentry-native

> SDK hooks signal handler

> enumerate loaded images

> dump state to disk before crash

   - stack walk with libunwindstack

SENTRY

SDK side

sentry-android

> watches file system for new events

> deserializes them, enhances them and uploads

SENTRY

Server side

> process crash reports
  - symbolicate native stacks on symbolicator
  - check for well known symbols in our buckets
  - resolve proguard for java stacks

> store

SENTRY

Shipping It

SENTRY

Android Gradle Plugin :'(

SENTRY

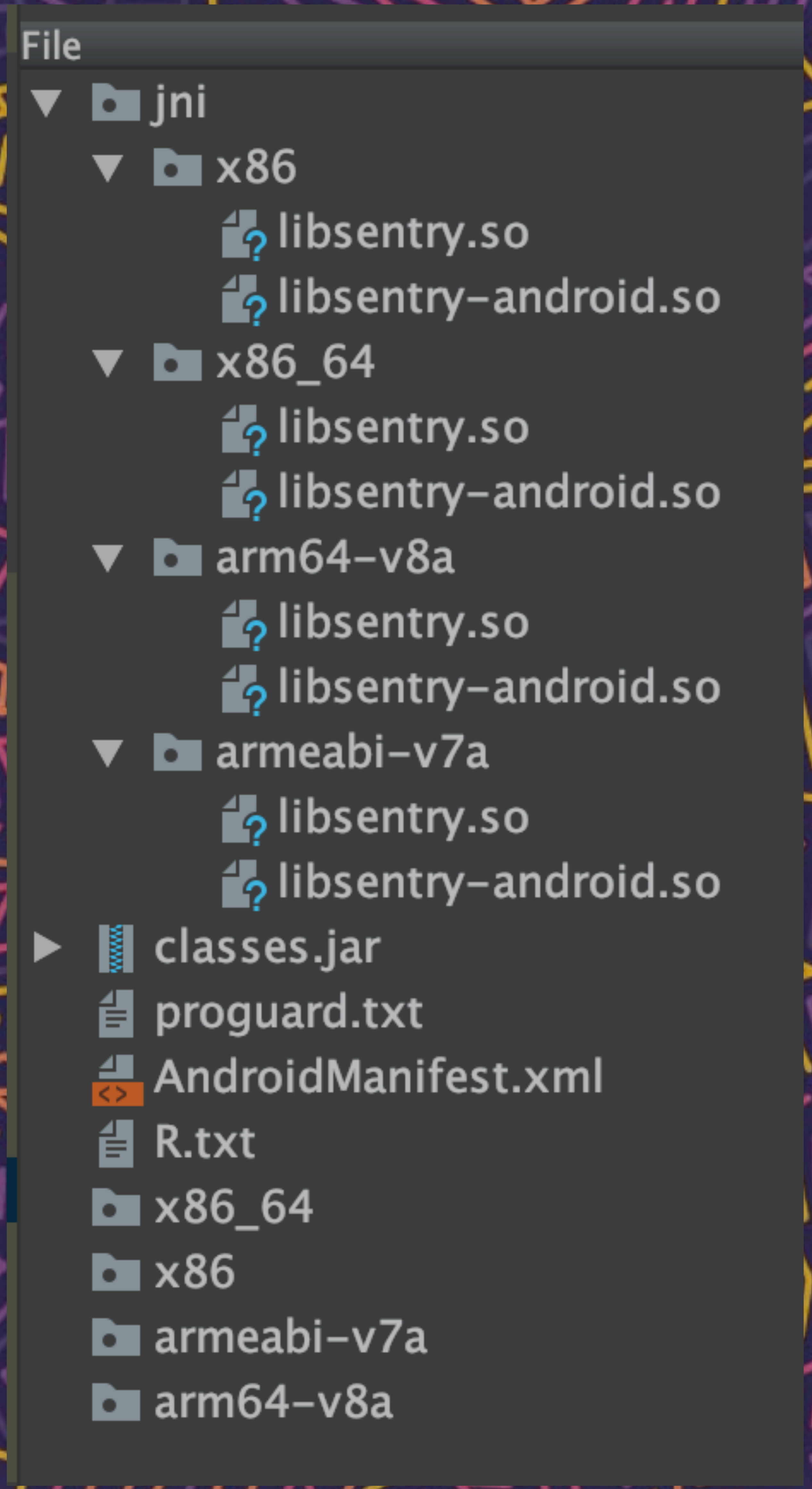
Structure

> cmake builds libraries per platform
  - these end up in folders for each architecture

where do the headers go?

how do we link to the libraries?

SENTRY



```
File
▼ jni
  ▼ x86
      libsentry.so
      libsentry-android.so
  ▼ x86_64
      libsentry.so
      libsentry-android.so
  ▼ arm64-v8a
      libsentry.so
      libsentry-android.so
  ▼ armeabi-v7a
      libsentry.so
      libsentry-android.so
▶ classes.jar
  proguard.txt
  AndroidManifest.xml
  R.txt
  x86_64
  x86
  armeabi-v7a
  arm64-v8a
```

Do The Ugly Dance

> needs a gradle plugin to
  - copy header libs out of AAR :(
  - so that code can link against the native lib

github.com/android/ndk-samples/issues/261
https://github.com/android/ndk/issues/916

Improving It

SENTRY

NDK asks

> a maintained and included stack walker
> make ucontext_t/getcontext available
> add support for shipping libs/headers in AARs
> Have OEMs/Google provide symbol servers

Q&A

SENTRY

sentry.io / @getsentry / @mitsuhiko / @brungarc